Red Hat OpenShift 上での NetCOBOL アプリケーション

開発・運用環境の構築手順

2019/02/08

富士通株式会社

1. はじめに

本資料では、NetCOBOL運用環境および NetCOBOL開発・運用環境をRed Hat OpenShift V3 上に構築する手順について説明します。

1.1 対象製品

本資料の対象製品は以下です。

- NetCOBOL Enterprise Edition 運用パッケージ V12.1.0 (64bit)
- NetCOBOL Enterprise Edition 開発・運用パッケージ V12.1.0 (64bit)

1.2 必要環境・媒体

以下の環境、媒体が必要です。

- Red Hat Enterprise Linux 7
- Red Hat OpenShift Container Platform （Red Hat OpenShift Online は対象外）
- NetCOBOL Enterprise Edition 運用パッケージまたは開発・運用パッケージのDVD

また、配備するアプリケーションに合わせた記憶領域の空き容量が必要です。

NetCOBOL Enterprise Edition 運用パッケージDVD のサイズは約340MB、アプリケーション未配備のDocker イメージのサイズは約780MB です。

NetCOBOL Enterprise Edition 開発・運用パッケージDVD のサイズは約380MB、アプリケーション未配備のDocker イメージのサイズは約1.1GB です。

1.3 検証環境

本資料の手順は以下を使用して検証しています。

- Red Hat Enterprise Linux 7.5
- Red Hat OpenShift Container Platform 3.9
- NetCOBOL Enterprise Edition 運用パッケージ V12.1.0 (64bit)
- NetCOBOL Enterprise Edition 開発・運用パッケージ V12.1.0 (64bit)

1.4. 前提知識

本資料を読む場合、以下の知識が必要です。

- Red Hat Enterprise Linux に関する基本的な知識
- Docker に関する基本的な知識
- Red Hat OpenShift に関する基本的な知識
- NetCOBOL に関する基本的な知識

1.5. 商標

2. 概要

Red Hat OpenShift 上での、NetCOBOL運用環境および NetCOBOL開発・運用環境の構築手順を、以下の3パターンに分けて説明します。

- NetCOBOL開発環境構築(ビルドコマンド実行方式)
- NetCOBOL開発環境構築(リモート開発方式)
- NetCOBOL運用環境構築

各手順について、コマンドの実行例およびファイルの作成例を記載しています。これらは本資料の手順を再現するための例として使用できますが、コマンドの引数やファイルに記載するパラメタは検証する環境に応じて適宜修正が必要です。
特に注意が必要な箇所については赤字で記しています。

3. NetCOBOL 開発環境構築(ビルドコマンド実行方式)

3.1. NetCOBOL の開発・運用パッケージ の Docker イメージ作成

以下の手順を実施して、NetCOBOL の開発・運用パッケージ のDocker イメージを作成します。
本手順はroot ユーザで実施してください。

1. NetCOBOL の開発・運用パッケージDVD のマウント
2.ベアイメージ（COBOL開発資産未配備）のDocker イメージ作成
3. NetCOBOL の開発・運用パッケージDVD のアンマウント
4. COBOL 開発資産を配備した Docker イメージ作成

3.1.1. NetCOBOL の開発・運用パッケージ DVD のマウント

NetCOBOL の開発・運用パッケージDVD をローカルディスクのディレクトリにマウントします。
以下は、ローカルディスクの/docker/netcobolimage にマウントする例です。

1. ローカルディスクにマウント先のディレクトリを作成します。

```
# mkdir -p /docker/netcobolimage
```

2. パッケージDVD をマウントします。

```
# mount -t iso9660 -r /dev/cdrom /docker/netcobolimage
```

3.1.2. ベアイメージ（COBOL 開発資産未配備）の Docker イメージ作成

NetCOBOLパッケージのインストールを行うためのDockerfile を作成します。
このファイルは、3.1.1 でNetCOBOL の運用パッケージのDVD をマウントしたディレクトリと同じ階層に配置します。

以下は、ベースイメージにregistry.access.redhat.com/rhel7.5を使用する例です。
図1: Dockerfile の例

```
FROM registry.access.redhat.com/rhel7.5
COPY netcobolimage /tmp/isomnt
RUN   /tmp/isomnt/INSTALL.sh -s
```

以下のコマンドを実施して、Docker イメージを作成します。
以下は、3.1.1 でNetCOBOL の運用パッケージのDVD を/docker/netcobolimage にマウントして、名前"netcoboleeimage "のDocker イメージを作成する例です。

```
# cd /docker
# docker build -t netcoboleeimage .
```

3.1.3. NetCOBOL の開発・運用パッケージ DVD のアンマウント

NetCOBOL の開発・運用パッケージDVD をローカルディスクのディレクトリからアンマウントします。

```
# umount /docker/netcobolimage
```

3.1.4. COBOL 開発資産を配備した Docker イメージ作成

コンテナ起動時に実行するシェルスクリプト"start.sh"と、シェルスクリプト内でビルドするCOBOL開発資産を準備します。

図2のシェルスクリプトを作成します。

図2:コンテナ起動時に実行するシェルスクリプトの例

・start.sh

```
#!/bin/bash

# COBOL開発環境の初期設定
source /opt/FJSVcbl64/config/cobol.sh
# COBOLアプリケーション”hellocobol”をビルド
cobol -dy -M -o hellocobol HelloCOBOL.cob
```

COBOL開発資産を準備します。今回の例では、標準出力にHello!　COBOLと出力するアプリケーション" hellocobol "を利用します。図3にソースコードを示します

図3: hellocobolのソースコード

・HelloCOBOL.cob

```
000010* Copyright 2019 FUJITSU LIMITED
000020 IDENTIFICATION DIVISION.
000030    PROGRAM-ID. HELLOCOBOL.
000040 DATA DIVISION.
000050    WORKING-STORAGE SECTION.
000060 PROCEDURE DIVISION.
000070        DISPLAY "Hello! COBOL"
000080 END PROGRAM HELLOCOBOL.
```

図4 の内容のDockerfile を作成します。

ベースイメージには、3.1.2.で作成したイメージを使用します。

図4: Dockerfile の例

```
FROM netcoboleeimage

#環境設定
RUN yum clean all -y
RUN rm -f /etc/rpm/macros.image-language-conf && ¥
      sed -i '/^override_install_langs=/d' /etc/yum.conf && ¥
```

```
    yum -y reinstall glibc-common
ENV LANG=ja_JP.UTF-8 ¥
    LANGUAGE="ja_JP:ja" ¥
    LC_ALL="ja_JP.UTF-8"
RUN yum -y reinstall tzdata && ¥
    yum -y install yum-langpacks system-config-language && ¥
    ln -snf /usr/share/zoneinfo/Asia/Tokyo /etc/localtime && ¥
    sed -ri 's/en_US/ja_JP/' /etc/locale.conf

# 開発に必要なパッケージのインストール
RUN yum install -y ¥
        gcc ¥
        gdb ¥
        imake ¥
        make ¥
        openssl ¥
        tcsh ¥
        glibc               glibc.i686 ¥
        elfutils-libelf elfutils-libelf.i686 ¥
        libstdc++           libstdc++.i686

WORKDIR /home/netcobol
RUN chmod ugo+rw /home/netcobol

# COBOL開発資産のコピー
COPY HelloCOBOL.cob HelloCOBOL.cob
RUN chmod ugo+rw HelloCOBOL.cob

# 起動スクリプトのコピー
COPY start.sh start.sh
RUN chmod ugo+rx start.sh

CMD ["/home/netcobol/start.sh"]
```

Docker イメージを作成します。
Dockerfile、実行するシェルスクリプト"start.sh"、シェルスクリプト内でビルドする

COBOL開発資産を同一ディレクトリに配置します。以下は、ディレクトリ"/docker/myapp"に配置して、名前"netcobolmyappbuild "のDocker イメージを作成する例です。

```
# cd /docker/myapp
# docker build -t netcobolmyappbuild .
```

Docker イメージからDocker コンテナを起動します。
以下はDocker イメージ" netcobolmyappbuild" からDocker コンテナ"netcobolcontainer"を起動する例です。

```
# docker run -it --name netcobolcontainer netcobolmyappbuild
```

コンテナの起動に成功すると、以下の文字がコンソールに出力されます。

```
最大重大度コードは I で，翻訳したプログラム数は 1 本です.
```

3.1.5. ビルド成果物の永続化

コンテナ内でビルドした成果物は、コンテナが破棄されると削除されます。
成果物を保持したい場合、以下のいずれかの方法でデータを永続化させてください。

a. ビルド成果物を永続ボリュームへコピーする

シェルスクリプト“start.sh”を図5のように修正し、ビルド成果物を永続ボリューム”/home/data”にコピーします。
図5:修正したシェルスクリプトの例
・start.sh

```
#!/bin/bash

# COBOL開発環境の初期設定
source /opt/FJSVcbl64/config/cobol.sh
# COBOLアプリケーション”hellocobol”をビルド
cobol -dy -M -o hellocobol HelloCOBOL.cob
#  ビルド成果物を永続ボリューム”/home/data”にコピー
cp hellocobol /home/data
```

以下はホストの” /docker/myapp/data”を永続ボリューム”/home/data”として割り当てて起動する例です。

```
# docker run -it --privileged -v /docker/myapp/data:/home/data --name netcobolcontainer
netcobolmyappbuild
```

b. COBOL 開発資産を永続ボリュームに格納し、コンテナから参照する

シェルスクリプト“start.sh”を図6のように修正し、COBOL開発資産を永続ボリューム”/home/data”に格納した状態でビルドを行います。

図6:修正したシェルスクリプトの例

・start.sh

```
#!/bin/bash

# COBOL開発環境の初期設定
source /opt/FJSVcbl64/config/cobol.sh
# COBOLアプリケーション”hellocobol”をビルド
cd /home/data
cobol -dy -M -o hellocobol HelloCOBOL.cob
```

以下はホストの” /docker/myapp/data”を永続ボリューム”/home/data”として割り当てて起動する例です。

```
# docker run -it --privileged -v /docker/myapp/data:/home/data --name netcobolcontainer
netcobolmyappbuild
```

3.2. Red Hat OpenShift 上での NetCOBOL アプリケーションのビルド

以下の手順を実施して、COBOLアプリケーション をRed Hat OpenShiftでビルドします。

本手順は、Docker イメージを作成した環境で実行します。

1. シェルスクリプトを修正します。
2. Docker イメージを再作成します。
3. Red Hat OpenShift からアクセス可能なリポジトリに、Docker イメージを登録します。

3.2.1.シェルスクリプトの修正

シェルスクリプト“start.sh”を図7のように修正し、コンテナが起動し続けるようにします。

図7:修正したシェルスクリプトの例

・start.sh

```
#!/bin/bash
source /opt/FJSVcbl64/config/cobol.sh
cobol -dy -M -o hellocobol HelloCOBOL.cob

while :
do
        sleep 1
done
```

3.2.2. Docker イメージを再作成

修正を反映するために、Docker イメージを再作成します。

```
# docker build -t netcobolmyappbuild .
```

3.2.3 Red Hat OpenShift からアクセス可能なリポジトリに Docker イメージを登録

1.Openshiftからアクセス可能なリポジトリに、Dockerイメージを登録します。

以下は、ローカルリポジトリ"192.168.100.102:5000" に、Docker イメージを"192.168.100.102:5000/netcobolmyappbuild:latest"として登録する例です。

```
# docker tag netcobolmyappbuild:latest
192.168.100.102:5000/netcobolmyappbuild:latest
# docker push 192.168.100.102:5000/netcobolmyappbuild:latest
```

2. Red Hat OpenShift にログインします。

3. Red Hat OpenShift 上で、リポジトリに登録したDocker イメージを実行します。

```
# oc new-app -i netcobolmyappbuild:latest
```

4.コンテナの起動時にcobolアプリケーションが実行されたことを確認するために、podの出力ログを確認します。以下は、pod名が" po/netcobolmyappbuild-1-qhnl9"の場合の例です。

```
# oc logs po/netcobolmyappbuild-1-qhnl9
```

今回の例では、以下のように出力され、COBOLアプリケーションがビルドされたことが確認できます。

```
最大重大度コードは I で，翻訳したプログラム数は 1 本です.
```

3.2.4. ビルド成果物の永続化

コンテナ内でビルドした成果物は、コンテナが破棄されると削除されます。
成果物を保持したい場合、「3.1.5. ビルド成果物の永続化」の方法を参照してデータを永続化させてください。
ただし、コンテナに永続ボリュームを割り当てる方法については、Red Hat OpenShift でコンテナを起動する場合と、docker コマンドでコンテナを起動する場合で異なります。
Red Hat OpenShift のコンテナに永続ボリュームをリンクする方法は以下を参照してください。
Red Hat OpenShift Documentation(https://docs.openshift.com/)
OpenShift Container Platform
/ Developer Guide
/ Using Persistent Volumes

4. NetCOBOL 開発環境構築(リモート開発方式)

4.1. NetCOBOL の開発・運用パッケージ の Docker イメージ作成

以下の手順を実施して、NetCOBOL の開発・運用パッケージ のDocker イメージを作成します。
本手順はroot ユーザで実施してください。

1. NetCOBOL の開発・運用パッケージDVD のマウント
2.ベアイメージ（COBOL開発資産未配備）のDocker イメージ作成
3. NetCOBOL の開発・運用パッケージDVD のアンマウント
4. COBOL 開発資産を配備した Docker イメージ作成

手順 1～3 のベアイメージ作成方法は「3. NetCOBOL 開発環境構築(ビルドコマンド実行方式)」と共通です。本章では、上記の章で作成したベアイメージ”netcoboleeimage”を使用します。
また、シェルスクリプト内でビルドする COBOL 開発資産は、「3.1.4. COBOL 開発資産を配備した Docker イメージ作成」で使用した”HelloCOBOL.cob”を使用します。

4.1.1. COBOL 開発資産を配備した Docker イメージ作成

NetCOBOLのリモート開発をDockerコンテナ上で使用する場合には、systemdターゲットを取得するためにsystemctlコマンドが正常動作することが条件となります。
しかし、通常のコンテナ起動モードではsystemctlコマンドが動作しないため(※1)、NetCOBOLのリモート開発はできません。

よって、リモート開発時にはsystemctlコマンドを正常動作させるためにコンテナを特権モード(※2)で起動する必要があります。

(※1)通常の起動モードでDockerコンテナを起動しsystemctlコマンドを実行すると
エラー(Failed to get D-Bus connection: No connection to service manager.)
となります。

(※2)特権モードは運用環境では非推奨のため、リモート開発時のみ使用してください。

図1 の内容のDockerfile を作成します。
ベースイメージには、3.1.2.で作成したイメージを使用します。
図1: Dockerfile の例

```
FROM netcoboleeimage

#環境設定
RUN yum clean all -y
RUN rm -f /etc/rpm/macros.image-language-conf && ¥
    sed -i '/^override_install_langs=/d' /etc/yum.conf && ¥
    yum -y reinstall glibc-common
ENV LANG=ja_JP.UTF-8 ¥
    LANGUAGE="ja_JP:ja" ¥
    LC_ALL="ja_JP.UTF-8"
RUN yum -y reinstall tzdata && ¥
    yum -y install yum-langpacks system-config-language && ¥
    ln -snf /usr/share/zoneinfo/Asia/Tokyo /etc/localtime && ¥
    sed -ri 's/en_US/ja_JP/' /etc/locale.conf

# 開発に必要なパッケージのインストール
RUN yum install -y ¥
        gcc ¥
        gdb ¥
        imake ¥
        make ¥
        openssl ¥
        tcsh ¥
```

```
        glibc               glibc.i686 ¥
        elfutils-libelf elfutils-libelf.i686 ¥
        libstdc++           libstdc++.i686

# リモート開発用デーモンの登録に必要なパッケージのインストール
RUN yum install -y ¥
        redhat-lsb-core ¥
        initscripts

# リモート開発用ポート設定
EXPOSE 61999
# リモートデバッグ用ポート設定
EXPOSE 59998

# 開発用ユーザの追加
ARG devuser
ARG devpass
RUN useradd ${devuser}
RUN echo "${devuser}:${devpass}" | chpasswd
RUN echo "source /opt/FJSVcbl64/config/cobol.sh" >> /home/${devuser}/.bashrc
WORKDIR /home/${devuser}
RUN chmod ugo+rw /home/${devuser}

# COBOL開発資産のコピー
COPY HelloCOBOL.cob HelloCOBOL.cob
RUN chmod ugo+rw HelloCOBOL.cob

CMD ["/sbin/init"]
```

Docker イメージを作成します。
DockerfileとCOBOL開発資産を同一ディレクトリに配置します。
以下は、ディレクトリ"/docker/myapp"に配置して、名前"netcobolremotebuild "のDocker イメージを作成する例です。
リモート開発を行うためのユーザ認証情報をdockerのbuild-argで指定します。

```
# cd /docker/myapp
# docker build -t netcobolremotebuild --build-arg devuser=開発用ユーザ名 --build-arg
```

```
devpass=開発用ユーザパスワード .
```

以下の説明では、開発用ユーザ名に”testuser”、 開発用ユーザパスワードに”testpass”を使用します。この場合、以下のコマンドを実行します。

```
# cd /docker/myapp
# docker build -t netcobolremotebuild --build-arg devuser=testuser --build-arg
devpass=testpass .
```

4.1.2. リモート開発サービスの起動

Docker イメージからDocker コンテナを起動します。
以下はDocker イメージ" netcobolremotebuild "からDocker コンテナ"netcobolcontainer" を起動する例です。NetCOBOLのリモート開発機能を使用するために、コンテナは特権モード(--privileged)で起動します。また、外部からコンテナにアクセスするためにポートフォワード(-p)を使用します。

```
# docker run -d --name netcobolcontainer --privileged -p 61999:61999 -p 59998:59998
netcobolremotebuild
```

以下のコマンドを実行し、NetCOBOLのリモート開発サービスを起動します。

```
# docker exec -it netcobolcontainer /opt/FJSVXrds/bin/enable-rds.sh
```

リモート開発サービスが正常に起動できたことを示すメッセージが表示されます。

```
Starting NetCOBOL Remote Development Service:                [  OK  ]
```

以下のコマンドを実行し、リモートデバッガコネクタをバックグラウンドで起動します。

```
# docker exec -d netcobolcontainer su testuser -c "source /opt/FJSVcbl64/config/cobol.sh
&& svdrds"
```

以下のコマンドを実行し、リモートデバッガコネクタが正常に起動できたこと確認します。

```
# docker exec -it netcobolcontainer ps aux | grep svdrds
```

リモートデバッガコネクタのプロセス情報が表示されます。
注: プロセス名(svdrds)以外の情報は、実行環境により異なります

```
testuser     8963   0.0   0.0     2912     868 ?          S      14:13     0:00 svdrds
```

NetCOBOL Studio からリモート開発を行う手順については、NetCOBOL Studio ユーザーズガイド 「第 9 章リモート開発」を参照してください。

4.1.3. ビルド成果物の永続化

ビルド成果物の永続化方法は、「3.1.5. ビルド成果物の永続化」を参照してください。

4.2. Red Hat OpenShift 上での NetCOBOL アプリケーションのリモート開発

以下の手順を実施して、COBOLアプリケーション をRed Hat OpenShiftでリモート開発します。本手順は、Docker イメージを作成した環境で実行します。

4.2.1 Red Hat OpenShift からアクセス可能なリポジトリに Docker イメージを登録

OpenShiftからアクセス可能なリポジトリに、Dockerイメージを登録します。
以下は、ローカルリポジトリ"192.168.100.102:5000" に、Docker イメージを"192.168.100.102:5000/netcobolremotebuild:latest"として登録する例です。

```
# docker tag netcobolremotebuild:latest
192.168.100.102:5000/netcobolremotebuild:latest
# docker push 192.168.100.102:5000/netcobolremotebuild:latest
```

4.2.2 Red Hat OpenShift 上でのリモート開発サービスの起動

以下の手順を実施して、NetCOBOLリモート開発サービスをOpenShift上で実行します。

参考
NetCOBOLリモート開発サービスを含むDockerコンテナは、root権限で特権コンテナとして実行する必要があります。root権限、かつ特権コンテナとしてDockerイメージを実行できるように、OpenShiftのユーザ、およびプロジェクトを設定してください。詳細は以下を参照してください。
Red Hat OpenShift Documentation(https://docs.openshift.com/)
OpenShift Container Platform
/ Cluster Administration
/ Managing Security Context Constraints

1. Red Hat OpenShift にログインします。

2. OpenShiftのNetCOBOLのテンプレートを作成します。
OpenShift上でリポジトリに登録したDockerイメージを取得してpod上で動くNetCOBOLのテンプレートを作成します。
以下の例では、netcobolremotebuild.yamlというファイル名のテンプレート(yaml形式)が出力されます。

```
# oc new-app -i netcobolremotebuild --name netcobolremotebuild -o yaml >
netcobolremotebuild.yaml
```

3. 2で出力されたテンプレートを編集して、以下の設定を行います。

－ NetCOBOLが動作するDockerコンテナを特権コンテナとして起動する。

"securityContext"と"livenessProbe"(以下の赤字の部分)を、DeploymentConfigの spec.template.spec.containersの中に追加します。

下記の例では、SCC(SECURITY CONTEXT CONSTRAINTS)が割り当てられたサービスアカウント“privsvcacct”を指定しています。インデントは以下の例に合わせてください。

```
--抜粋開始--
        spec:
          containers:
          - image: docker-registry.default.svc:5000/test04-prj/netcobolremotebuild:latest
            name: netcobolremotebuild
            ports:
            - containerPort: 59998
              protocol: TCP
            - containerPort: 61999
              protocol: TCP
            resources: {}
            securityContext:
              privileged: true
          securityContext:
            runAsUser: 0
          serviceAccount: privsvcacct
          serviceAccountName: privsvcacct
      test: false
--抜粋終了--
```

4. 3で編集したテンプレートを使ってコンテナを起動します。

```
# oc create -f netcobolremotebuild.yaml
```

5. コンテナのIP/PORTを公開します。

NetCOBOLリモート開発用のIPアドレスとポート番号を公開します。

以下は、pod名が” netcobolremotebuild-1-78fvk”の場合に、IPアドレス192.168.100.103として、ポート番号61999,59998を公開する例です。

```
# oc expose pod netcobolremotebuild-1-78fvk --port=61999,59998
--external-ip=192.168.100.103
```

以下のコマンドを実行し、コンテナに接続します。

```
# oc exec -it netcobolremotebuild-1-78fvk /bin/bash
```

以下のコマンドは、接続したコンテナ内で実行します。

NetCOBOLのリモート開発サービスを起動します。

```
# /opt/FJSVXrds/bin/enable-rds.sh
```

リモート開発サービスが正常に起動できたことを示すメッセージが表示されます。

```
Starting NetCOBOL Remote Development Service:                [  OK  ]
```

以下のコマンドを実行し、リモートデバッガコネクタを起動します。

```
# su testuser
$ source /opt/FJSVcbl64/config/cobol.sh
$ svdrds
```

リモートデバッガコネクタが正常に起動できたことを示すメッセージが表示されます。

```
x64 target Remote Debugger Connector
192.169.3.45:59998
Press Ctrl+C To End
```

NetCOBOL Studio からリモート開発を行う手順については、NetCOBOL Studio ユーザーズガイド 「第 9 章リモート開発」を参照してください。

4.2.3. ビルド成果物の永続化

ビルド成果物の永続化方法は、「3.2.4. ビルド成果物の永続化」を参照してください。

5. NetCOBOL 運用環境構築

5.1. NetCOBOL の運用パッケージ の Docker イメージ作成

以下の手順を実施して、NetCOBOL の運用パッケージ のDocker イメージを作成します。
本手順はroot ユーザで実施してください。

1. NetCOBOL の運用パッケージDVD のマウント

2.ベアイメージ（NetCOBOL アプリケーション未配備）のDocker イメージ作成
3. NetCOBOL の運用パッケージDVD のアンマウント
4. NetCOBOL アプリケーションを配備したDocker イメージ作成

5.1.1. NetCOBOL の運用パッケージ DVD のマウント

NetCOBOL の運用パッケージDVD をローカルディスクのディレクトリにマウントします。
以下は、ローカルディスクの/docker/netcobolimage にマウントする例です。

1. ローカルディスクにマウント先のディレクトリを作成します。

```
# mkdir -p /docker/netcobolimage
```

2. パッケージDVD をマウントします。

```
# mount -t iso9660 -r /dev/cdrom /docker/netcobolimage
```

5.1.2. ベアイメージ（NetCOBOL アプリケーション未配備）の Docker イメージ作成

NetCOBOLパッケージのインストールを行うためのDockerfile を作成します。
このファイルは、5.1.1 でNetCOBOL の運用パッケージのDVD をマウントしたディレクトリ同じ階層に配置します。

以下は、ベースイメージにregistry.access.redhat.com/rhel7.5を使用する例です。
図1: Dockerfile の例

```
FROM registry.access.redhat.com/rhel7.5
COPY netcobolimage /tmp/isomnt
RUN  /tmp/isomnt/INSTALL.sh -s
```

以下のコマンドを実施して、Docker イメージを作成します。
以下は、5.1.1 でNetCOBOL の運用パッケージのDVD を/docker/netcobolimage にマウントして、名前"netcoboleerimage "のDocker イメージを作成する例です。

```
# cd /docker
# docker build -t netcoboleerimage .
```

5.1.3. NetCOBOL の運用パッケージ DVD のアンマウント

NetCOBOL の運用パッケージDVD をローカルディスクのディレクトリからアンマウントします。

```
# umount /docker/netcobolimage
```

5.1.4. NetCOBOL アプリケーションを配備した Docker イメージ作成

コンテナ起動時に実行するシェルスクリプト"start.sh"と、シェルスクリプト内で起動するCOBOLアプリケーションを準備します。
本章では、COBOLアプリケーションの開発手順は解説を行いません。
以下では、「3. NetCOBOL開発環境構築(ビルドコマンド実行方式)」で作成したCOBOLアプリケーション”hellocobol”を使用します。

図2のシェルスクリプトを作成します。
図2:コンテナ起動時に実行するシェルスクリプトの例
・start.sh

```
#!/bin/bash

# COBOLランタイムの初期設定
source /opt/FJSVcbl64/config/cobolrt.sh
# COBOLアプリケーション”hellocobol”を実行
/home/netcobol/hellocobol
```

図3 の内容のDockerfile を作成します。
ベースイメージには、3.1.2.で作成したイメージを使用します。
図3: Dockerfile の例

```
FROM netcoboleerimage

#環境設定
RUN yum clean all -y
RUN rm -f /etc/rpm/macros.image-language-conf && ¥
    sed -i '/^override_install_langs=/d' /etc/yum.conf && ¥
    yum -y reinstall glibc-common
ENV LANG=ja_JP.UTF-8 ¥
    LANGUAGE="ja_JP:ja" ¥
    LC_ALL="ja_JP.UTF-8"
RUN yum -y reinstall tzdata && ¥
    yum -y install yum-langpacks system-config-language && ¥
    ln -snf /usr/share/zoneinfo/Asia/Tokyo /etc/localtime && ¥
    sed -ri 's/en_US/ja_JP/' /etc/locale.conf
```

```
WORKDIR /home/netcobol
RUN chmod ugo+rw /home/netcobol

# COBOLアプリケーションのコピー
COPY hellocobol hellocobol
RUN chmod ugo+rx hellocobol

# 起動スクリプトのコピー
COPY start.sh start.sh
RUN chmod ugo+rx start.sh

CMD ["/home/netcobol/start.sh"]
```

Docker イメージを作成します。
Dockerfile、実行するシェルスクリプト"start.sh"、シェルスクリプト内で起動するCOBOLアプリケーションを同一ディレクトリに配置します。以下は、ディレクトリ"/docker/myapp"に配置して、名前"netcobolmyapprun "のDocker イメージを作成する例です。

```
# cd /docker/myapp
# docker build -t netcobolmyapprun .
```

Docker イメージからDocker コンテナを起動します。
以下はDocker イメージ" netcobolmyapprun " からDocker コンテナ"netcobolcontainer"を起動する例です。

```
# docker run -it --name netcobolcontainer netcobolmyapprun
```

コンテナの起動に成功すると、以下の文字がコンソールに出力されます。

```
Hello! COBOL
```

5.2. Red Hat OpenShift 上での NetCOBOL アプリケーションの実行

以下の手順を実施して、COBOLアプリケーション をRed Hat OpenShiftで実行します。
本手順は、Docker イメージを作成した環境で実行します。
1. シェルスクリプトを修正します。
2. Docker イメージを再作成します。

3. Red Hat OpenShift からアクセス可能なリポジトリに、Docker イメージを登録します。

5.2.1.シェルスクリプトの修正

シェルスクリプト“start.sh”を図4のように修正し、コンテナが起動し続けるようにします。

図4:修正したシェルスクリプトの例

・start.sh

```
#!/bin/bash
source /opt/FJSVcbl64/config/cobolrt.sh
/home/netcobol/hellocobol

while :
do
        sleep 1
done
```

5.2.2. Docker イメージを再作成

修正を反映するために、Docker イメージを再作成します。

```
# docker build -t netcobolmyapprun .
```

5.2.3 Red Hat OpenShift からアクセス可能なリポジトリに Docker イメージを登録

1.Openshiftからアクセス可能なリポジトリに、Dockerイメージを登録します。

以下は、ローカルリポジトリ"192.168.100.102:5000" に、Docker イメージを"192.168.100.102:5000/netcobolmyapprun:latest"として登録する例です。

```
# docker tag netcobolmyapprun:latest 192.168.100.102:5000/netcobolmyapprun:latest
# docker push 192.168.100.102:5000/netcobolmyapprun:latest
```

2. Red Hat OpenShift にログインします。

3. Red Hat OpenShift 上で、リポジトリに登録したDocker イメージを実行します。

```
# oc new-app -i netcobolmyapprun:latest
```

4.コンテナの起動時にcobolアプリケーションが実行されたことを確認するために、podの出

力ログを確認します。以下は、pod名が” po/netcobolmyapprun-1-54jwj”の場合の例です。

```
# oc logs po/netcobolmyapprun-1-54jwj
```

今回の例では、以下のように出力され、COBOLアプリケーションが動作したことが確認できます。

```
Hello! COBOL
```

6. パッチ適用手順（UpdateAdvisor 使用法）

以下の手順を実施して、NetCOBOL 緊急修正を適用します。
本手順はroot ユーザで実施してください。

1. 修正適用に必要な資産をコンテナにコピーする
UpdateAdvisor のインストール資材、および緊急修正モジュールを、コンテナ上の任意の場所にコピーします。

2. 修正を適応する Docker コンテナにログインする
修正適用時には systemctl コマンドを正常動作させるためにコンテナを特権モードで起動し、コンテナで起動されるコマンドを「/sbin/init」に変更する必要があります。
以下のコマンドを実行して、修正を適応する Docker コンテナを特権モードで起動し、コンテナにログインします。
以下は、修正を適用するコンテナ名が” netcobolcontainer”の場合の例です。

```
# docker run -d --privileged --name netcobolcontainer netcoboleeimage /sbin/init
# docker exec -it netcobolcontainer /bin/bash
```

3. UpdateAdvisor をインストールする
UpdateAdvisor のインストール方法については UpdateAdvisor のヘルプを参照してください。

4. NetCOBOL の緊急修正を適用する
NetCOBOL の緊急修正の適用方法については、緊急修正の修正情報ファイル、および UpdateAdvisor のヘルプを参照してください。

付録 A. syslog の使用方法

コンテナの障害調査や動作検証で syslog を使用するための手順について説明します。

1. Dockerfile を修正し、syslogd をインストールする

図1 の内容を、syslogを使用するコンテナ用のDockerfile に追記します。

図1: Dockerfile の追記例

```
# Install syslogd
RUN yum install -y rsyslog
RUN sed 's/$ModLoad imjournal/# $ModLoad imjournal/' -i /etc/rsyslog.conf && ¥
    sed 's/$OmitLocalLogging on/$OmitLocalLogging off/' -i /etc/rsyslog.conf && ¥
    sed 's/$IMJournalStateFile imjournal.state/# $IMJournalStateFile
imjournal.state/' -i /etc/rsyslog.conf
```

2. Docker イメージを再作成

修正を反映するために、Docker イメージを再作成します。

3. Docker イメージから Docker コンテナを起動

修正を反映するために再作成した Docker イメージから Docker コンテナを起動します。

4. コンテナに接続

5. rsyslogdを起動

以下のコマンドを実行し、rsyslogdが正常に起動できたこと確認します。

```
# ps aux | grep rsyslogd
```

rsyslogdのプロセス情報が表示されます。

注: プロセス名(rsyslogd)以外の情報は、実行環境により異なります

```
root 8511  0.0  0.0 172336  1840 ?     Ssl  11:04     0:00 /usr/sbin/rsyslogd -n
```

rsyslogdが起動していない場合、以下のコマンドを実行し、rsyslogdを起動します。

```
# rsyslogd
```

以上